Timbre Explorer Documentation

Release 0.0.1

Doug Olson

October 22, 2016

CHAPTER

ONE

OVERVIEW

The Timbre Explorer module creates Timbre objects for the investigation of the dissonance and consonance properties of musical timbres. The disMeasure() function is a Python translation of William Sethares's matlab and C code. I have added timbre objects and functions to do plots of various kinds and to generate .wav files so you can hear the timbres and their dissonance / consonance patterns. Requires Matplotlib, Numpy and Scipy.

Objects can be initialized as:

- Even
- Odd
- Evenodd
- Square
- Sawtooth
- Triangle
- Beam
- Custom

Most objects have the following properties:

- **f_0**: the fundamental frequency. Default is 220 Hz
- **n_partials**: the desired number of partials in the tone. Default is 7
- **attenuation**: the attenuation rate to be applied to the partials. Default is .7071

Specific waveforms such as Square, Sawtooth etc, have predefined spectra and attenuation rates

The Custom object allows the user to specify any desired set of partials and amplitudes.

Typical usage, assuming you cd to the directory that contains the **Timbre** directory and run Python 2.7x from there:

```
>>> import Timbre
>>> foo = Timbre.Even() # Create an object with even partials and default parameters
>>> bar = Timbre.Beam(f_0 = 327, numPartials = 17, attenuation = .5) # Create an object with partial
>>> freqs = [300, 600, 900, 1200]
>>> amps = [1, .5, .3, .9]
>>> baz = Timbre.Custom(freqs, amps) # Create an object with custom partials
>>> foo.disPlot() # plot the dissonance curve for the timbre
>>> bar.ConsDisFreqs(makePlot = True) # plot identifies maximima and minima in the dissonance curve
>>> baz.partialsPlot() # bar plot of the relative amplitudes and frequencies for the partials of the
>>> foo.wavePlot() # plot one period of the timbre's waveform
>>> bar.timbreGen() # Generate a 5 second sample of the timbre
>>> baz.timbreSweep(length = 60) # Generate a sweep of the timbre against itself
>>> Timbre.disPlotMultiple(foo, bar, baz) # Generate a dissonance plot for timbres foo, bar and baz
>>> foo.writeConsonantChord() # Writes .wav chord, user selected intervals, based on consonant freque
>>> foo.writeEqTempChord() # Writes .wav chord, user selected intervals, based on equal tempered frec
```

CHAPTER

TWO

TIMBRE.TIMBRE MODULE

class `Timbre.timbre.`**`Timbre`**(*f_0*, *numPartials*, *octaves=1*)

Bases: `object`

Parent class for the various timbre objects. Use subclasses Even, Odd, Evenodd, Square, Sawtooth, Triangle, Beam and Custom to create objects

`audioGenPath`()

Creates a directory for audio files to be written to.

`consDisFreqs`(*makePlot=False*)

Generates a list of the consonances (minima on the dissonance plot) for a given timbre. The consonances are used by consFreqs and writeChord. Optionally plots a bar graph showing the peak dissonant an consonant frequencies for a given Timbre object

`consFreqs`(*makePlot=False*)

Generates an array of consonant frequencies for a given Timbre. Optionally plots a bar graph showing the peak dissonant an consonant frequencies for a given Timbre object

`disMeasure`(*octaves=1*)

Calculates the relative dissonance for a given Timbre object at all intervals within a specified range. disMeasure(octaves = 1).

returns an an array of relative dissonances **self.dissonances** and a normalized dissonance array **self.norm**

`disPlot`(*normalized=True*)

Plots relative dissonance at all intervals for a given tone

`partialsPlot`()

Plots the partials and their relative amplitudes for the timbre being examined. Positive Amplitudes in red, Negative amplitudes in blue.

`timbreGen`()

Generates audio data and a .wav file of a timbre. * prompts user for file length * files in TimbreAudio in cwd

`timbreSweep`()

Generates audio data and a .wav file of a timbre swept against itself, i.e. one tone is held constant, the other ascends for just over 1 octave. * prompts user for file length * files in TimbreAudio in cwd

`wavePlot`()

Generates a plot of the waveform for the timbre being examined

`writeConsonantChord`(*verbose=False*)

Writes a chord with the selected half steps at the timbre's nearest consonant pitches.

- Prompts user for length and desired half steps.

- Has some trouble with Odd and Beam timbres

`writeEqTempChord()`

Writes an equal tempered chord with the selected half steps.

- Prompts user for length and desired half steps.

CHAPTER

THREE

TIMBRE.GENERATORS MODULE

class `Timbre.generators.`**`Even`** (*f_0=220*, *numPartials=7*, *attenuation=0.7071*)

Bases: *`Timbre.timbre.Timbre`*

A Timbre object with Even harmonics.

class `Timbre.generators.`**`Odd`** (*f_0=220*, *numPartials=7*, *attenuation=0.7071*)

Bases: *`Timbre.timbre.Timbre`*

A Timbre object with Odd harmonics

class `Timbre.generators.`**`Evenodd`** (*f_0=220*, *numPartials=7*, *attenuation=0.7071*)

Bases: *`Timbre.timbre.Timbre`*

A Timbre object with Even and Odd harmonics

class `Timbre.generators.`**`Square`** (*f_0=220*, *numPartials=7*)

Bases: *`Timbre.timbre.Timbre`*

A Timbre object with A Square wave (additive synthesis). Odd Harmonics. Attenuation rate is fixed at 1 / n

class `Timbre.generators.`**`Sawtooth`** (*f_0=220*, *numPartials=7*)

Bases: *`Timbre.timbre.Timbre`*

A Timbre object with A Sawtooth wave (additive synthesis). Attenuation rate is fixed at 2*(-1^n)/n * pi, n being the partial #

class `Timbre.generators.`**`Triangle`** (*f_0=220*, *numPartials=7*)

Bases: *`Timbre.timbre.Timbre`*

A Timbre object with A Triangle wave (additive synthesis). Attenuation rate is fixed at 0.8105694691387022*((-1)^((n - 1)/2.)/(n^2)) n being the partial #

class `Timbre.generators.`**`Custom`** (*freq_array*, *amps_array*)

Bases: *`Timbre.timbre.Timbre`*

A Timbre object with Custom harmonics.

- freq_array: an array of frequencies
- amps_array: an array of amplitides
- freq_array and amps_array must have the same length

class `Timbre.generators.`**`Beam`** (*f_0=220*, *numPartials=7*, *attenuation=0.7071*)

Bases: *`Timbre.timbre.Timbre`*

A Timbre object with Beam harmonics. Partial frequencies are .4413 * f_0*(n +.5)^2

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX